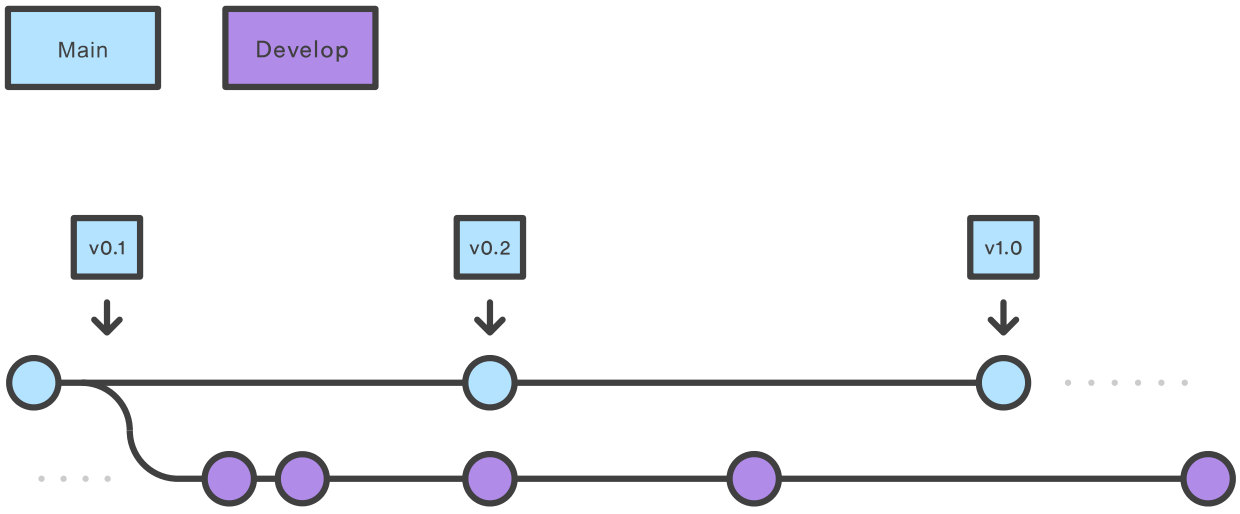
Flujo de trabajo en los repositorios

Como flujo principal en la administración de los repositorios de los proyectos desarrollados en la UPRA, se escogió el flujo definido como GitFlow.

GitFlow

Gitflow es un modelo de flujo de trabajo para el control de versiones utilizando Git, diseñado para proyectos de software que requieren un ciclo de vida de desarrollo más estructurado. Fue propuesto por Vincent Driessen en 2010 y se ha convertido en un enfoque popular para la gestión de ramas y versiones en proyectos de desarrollo de software.



Principales características de Gitflow:

1. **Ramificación por convención:**

- Gitflow define una estructura de ramificación clara y predecible. Utiliza diferentes ramas para diferentes tipos de trabajo, lo que facilita la organización y seguimiento del progreso del desarrollo.

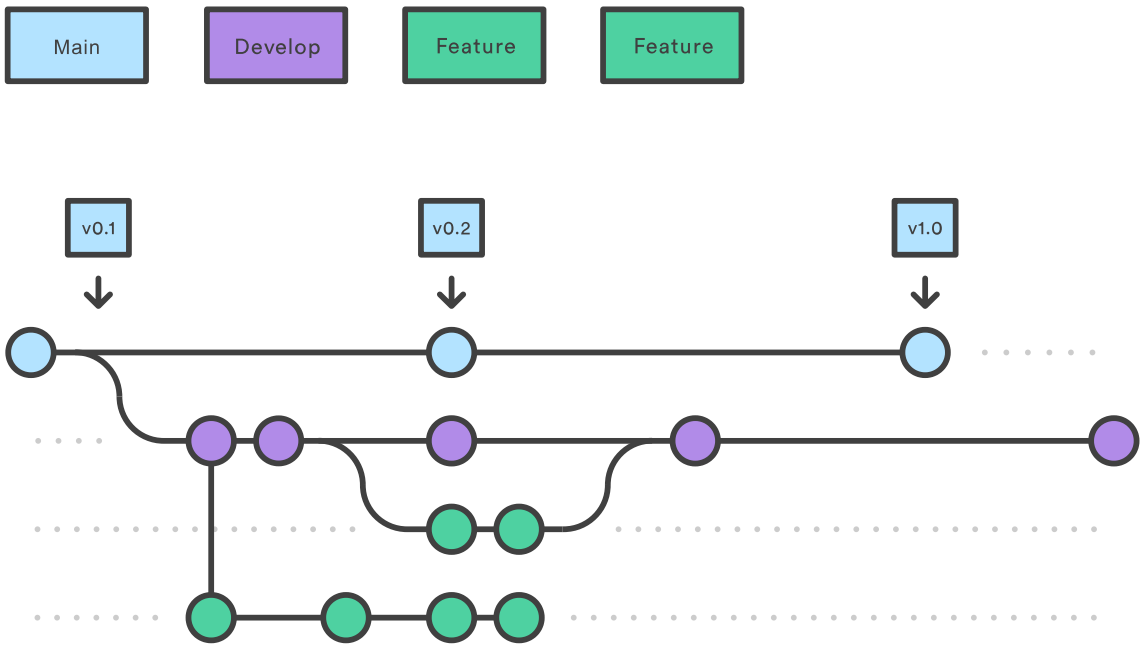
2. **Separación de roles:**
    - Gitflow separa las tareas de desarrollo de características, mantenimiento de versiones y resolución de problemas críticos en ramas específicas. Esto permite una mejor gestión de cada etapa del ciclo de vida del software.
3. **Versionado semántico:**
    - Gitflow fomenta el uso de versionado semántico, donde los números de versión tienen un significado claro y predecible, lo que facilita la identificación de cambios y la comunicación entre los miembros del equipo.

Componentes principales:

Ramas principales:

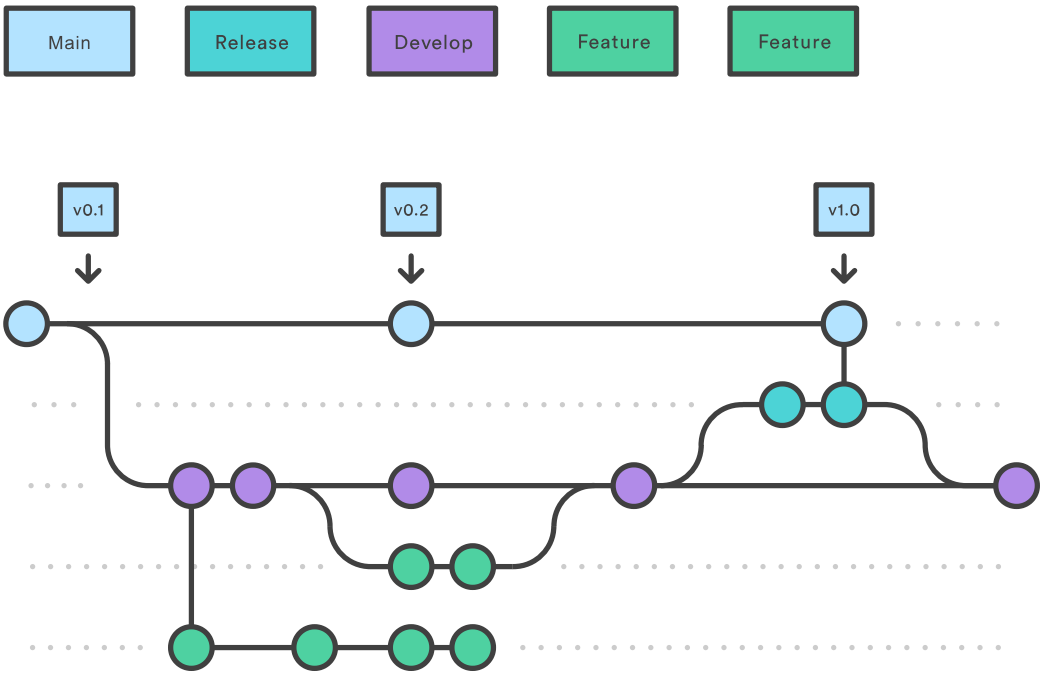
- **Master:** Esta rama representa la versión estable y desplegable del software. Es común que solo se acepten cambios en esta rama a través de merges de otras ramas, como `release` y `hotfix`.
- **Develop:** Es la rama de integración principal. Todos los desarrollos de características se fusionan en esta rama. Es una rama en constante evolución y se espera que contenga el código más reciente y funcional del proyecto.

Ramas de funciones:



- **Feature:** Cada nueva característica o mejora se desarrolla en su propia rama de característica, que se bifurca de `develop`. Estas ramas son independientes y se pueden desarrollar simultáneamente por diferentes miembros del equipo. Una vez que la característica está completa y probada, se fusiona nuevamente en `develop`.

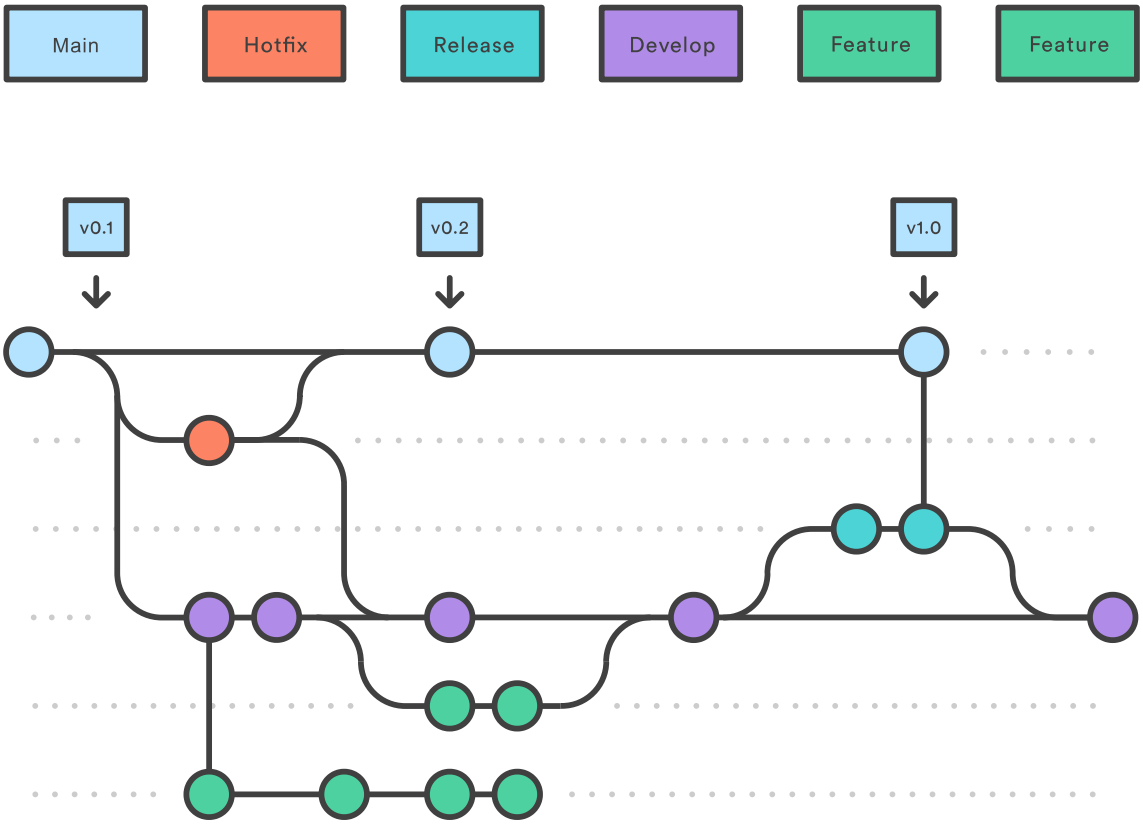
Ramas de lanzamiento:



- **Release:** Cuando `develop` ha acumulado suficientes características nuevas y está listo para una nueva versión, se crea una rama de lanzamiento a partir de `develop`. En esta rama, se realizan pruebas finales y ajustes menores antes del lanzamiento. Una vez que está lista para ser lanzada, se fusiona en `master` y `develop`, y se etiqueta con el número de versión.

Branches de corrección:

Las ramas de corrección (ya sea hotfix o bugfix) siguen un proceso similar: se crean para abordar un problema específico, se resuelve el problema en la rama correspondiente y luego se fusiona de vuelta en la rama principal adecuada una vez que se ha probado y validado la solución.



- **Hotfix:** Si se descubre un error crítico en producción, se crea una rama de hotfix a partir de `master`. Se corrige el error en esta rama y luego se fusiona en `master` y `develop`, asegurando que el cambio también se refleje en el próximo lanzamiento.
- **Bugfix:** las ramas de "bugfix" o ramas de corrección de errores. Funcionan de manera similar a las ramas de hotfix, pero se centran en problemas que no tienen un impacto crítico en la funcionalidad o estabilidad del producto. Estas ramas pueden crearse a partir de `master` o `develop`, y una vez que se resuelve el error, se fusionan en `develop`.

Políticas a implementarse en los repositorios:

Para aplicar el uso de GitFlow en el desarrollo de aplicaciones en la UPRA se deben establecer las siguientes políticas en todos los repositorios de los

diferentes proyectos de la UPRA.

Política de Ramificación:

Se debe limitar quién puede crear nuevas ramas (solo los desarrolladores y/o administradores del proyecto) y desde qué ramas pueden crearse para mantener un control adecuado sobre la estructura del repositorio.

Se definieron las siguientes convenciones de nomenclatura para las ramas para mantener la consistencia y facilitar la identificación de su propósito.

Ramas principales:

Las ramas principales serán las ramas `main` o `master` y `develop`.

Para los proyectos nuevos, se recomienda utilizar la rama `main` como la rama principal en lugar de `master`. Esto se debe a que algunas comunidades consideran que el término `master` puede ser ofensivo (https://sfconservancy.org/news/2020/jun/23/gitbranchname/). Sin embargo, si no es posible cambiar el nombre, la rama `master` seguirá siendo considerada como la rama principal para proyectos antiguos.

`main` o `master`	Contiene el código estable y desplegable en producción
`develop`	Rama de integración principal donde se fusionan todas las características nuevas.

Ramas de funciones:

Estas ramas son utilizadas para la implementación de nuevas funcionalidades. Deben ser generadas a partir de `develop`.

Por convención en nombre debe iniciar con `feature/`, por ejemplo: `feature/nombre-funcionalidad`.

Ramas de lanzamiento:

Se utilizan para preparar una versión para el lanzamiento final. Se crean a partir de `develop` y contienen cambios destinados a la próxima versión.

Por convención en nombre debe iniciar con `release/` y finalizar con el nombre de versión a desplegar, por ejemplo: `release/1.0.0`. Pueden generarse versiones de prueba, como versiones alpha, beta o release candidates.

También se recomienda crear un tag con en nombre de versión cuando se cree una rama de lanzamiento.

Ramas de error:

Se utilizan para corregir problemas de la aplicación. Existen dos tipos de ramas de error. Las ramas `hotfix` y las ramas `bugfix` .

Para las ramas `hotfix` el nombre deber iniciar con `hotfix/`, por ejemplo: `hotfix/correccion-error-critico`. Estas ramas deben ser mezcladas tanto en `develop` como en `main`.

Para las ramas `fix` el nombre deber iniciar con `fix/`, por ejemplo: `fix/correccion-error`. Estas ramas deben mezclarse en la rama `develop` y se desplegarán en el proximo lanzamiento.

Política de Protección de Rama:

Se debe proteger las ramas principales (`main` , `master`, `develop`) para evitar cambios directos y requerir que todas las modificaciones se realicen a través de pull requests.

Política de Revisión de Código:

Para todos los proyectos se debe requerir al menos una revisión de código antes de que se puedan fusionar los cambios en la rama principal. Esto garantiza que todos los cambios sean revisados por al menos un par de ojos antes de su incorporación en la ramas principales.

Según sean los requerimientos del proyecto, se pueden configurar políticas de revisión de código para requerir aprobaciones de revisores específicos o de un

grupo de revisores designado.

Política de Pruebas Automáticas:

Según la naturaleza de los proyectos, donde sea posible, se deben configurar flujos de trabajo de compilación y pruebas automáticas que se ejecuten en todas los pull requests para garantizar que el código cumpla con los estándares de calidad.

Para que un pull request pueda ser aprobado, debe haber superado todas las pruebas automáticas definidas para el proyecto.

Consideraciones generales

- Se recomienda borrar las ramas de error y de funciones una vez estas fueron mezcladas a las ramas principales.
- Se debe evitar reutilizar las ramas, una vez mezcladas a las ramas principales. Si se requiere corregir o ajustar algo, se recomienda generar una rama nueva.